Tentamen
EDAF85 – Realtidssystem

2022-08–25, 14.00–19.00

Det är tillåtet att använda Java snabbreferens och miniräknare. Det går bra att använda både engelska och svenska uttryck för svaren.

Den här tentamen består av två delar: *teori*, fråga 1-4 (15 poäng), och *programmeringsuppgifter*, fråga 5-6 (15 poäng). För godkänd (betyg 3) krävs preliminärt sammanlagt hälften av alla 30 möjliga poäng samt en tredjedel av poängen (5) på varje del för sig.

Formelsamling

$$\sum_{i=1}^{n} \frac{C_i}{T_i} \leq n(2^{1/n} - 1)$$

$$R_i = C_i + B_i + \sum_{j \in hp(i)} \left\lceil \frac{R_i}{T_j} \right\rceil C_j$$

1. Nedanstående Javaprogram har en tendens att hamna i dödläge.

```
1  public static void main(String[] args){
2
3      Lock lockA = new ReentrantLock();
4      Lock lockB = new ReentrantLock();
5      Lock lockC = new ReentrantLock();
6      Lock lockD = new ReentrantLock();
7      Lock lockE = new ReentrantLock();
8
9      new Thread(() -> {
10         while (true) {
11             lockD.lock();
12             useD();
13             lockE.lock();
14             lockA.lock();
15             useADE();
16             lockA.unlock();
17             lockE.unlock();
18             lockD.unlock();
19             lockB.lock();
20             lockC.lock();
21             useBC();
22             lockC.unlock();
23             lockB.unlock();
24         }
25     }).start();
26
27     new Thread(() -> {
28         while (true) {
29             lockE.lock();
30             useE();
31             lockE.unlock();
32             lockB.lock();
33             lockE.lock();
34             useBE();
35             lockB.unlock();
36             useE();
37             lockE.unlock();
38             lockA.lock();
39             lockB.lock();
40             useAB();
41             lockB.unlock();
42             lockA.unlock();
43         }
44     }).start();
45
46     while (true) {
47         lockA.lock();
48         useA();
49         lockA.unlock();
50         lockB.lock();
51         lockC.lock();
52         useBC();
53         lockD.lock();
54         useBCD();
55         lockD.unlock();
56         lockC.unlock();
57         lockB.unlock();
58     }
59 }
```

Anmärkningar

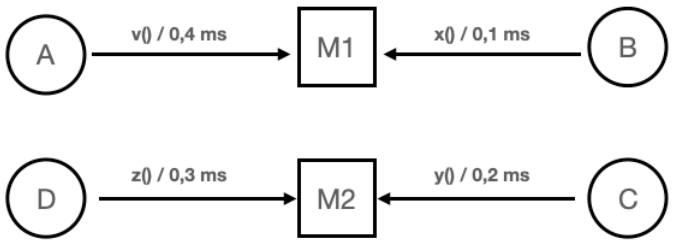
- Operationerna av typen `useXYZ()` har utelämnats, men är ej relevanta för uppgiften.
- `Runnable` är ett så kallat *funktionellt interface*. Det innebär att man som i exemplet ovan (och som vi sett i kursen) kan använda ett lambdauttryck som parameter till konstruktorn för klassen `Thread` för att deklarera och skapa en ny tråd.
- `Reentrantlock` representerar, som vi gått igenom i kursen, en mutexsemafor där operationen `lock()` låser resursen och `unlock()` frigör den.

a) Rita en resursallokeringsgraf för programmet ovan (ge lämpliga egna beteckningar för trådarna och resurserna). *(2p)*

b) Ange på vilka kodrader de olika trådarna måste befinna sig då dödläge inträffar. *(1,5p)*

c) Dödläge kan undvikas om vi byter plats på två kodrader i programmet. Vid varje anrop av operationer av typen `useXYZ()` ska samma lås vara låsta som tidigare. Vilka är dessa två rader? *(0,5p)*

2. Antag att vi har ett system som schemaläggs enligt Deadline Monotonic Scheduling, använder sig av dynamiskt prioritetsarv, och består av fyra trådar kallade A, B, C och D. Trådarna använder sig av två monitorer, kallade M1 och M2, för sin inbördes synkronisering enligt nedanstående figur där också respektive monitoroperations maximala exekveringstid är angiven.



Trådarnas värstafallsexekveringstider (C), periodtider (T) och deadlines (D) ges av följande tabell:

Tråd	*C (ms)*	*T (ms)*	*D (ms)*
A	4	20	20
B	1	4	2
C	3	12	6
D	2	24	12

a) Vi skulle kunna räkna ut det totala CPU-utnyttjandet för systemet, dvs $\sum_{i=1}^{n} \frac{C_i}{T_i}$. Vilken, om någon, slutsats kan vi dra om det aktuella systemets schemaläggningsbarhet utifrån värdet vi då får fram? Motivera ditt svar. *(1p)*

b) Räkna ut B_i för respektive tråd. *(2p)*

c) Räkna nu ut R_i för respektive tråd. För full poäng måste du redovisa alla dina beräkningar. *(2p)*

d) Är systemet schemaläggningsbart? För att få poäng krävs en korrekt motivering till svaret. *(1p)*

3. I samband med semaforer har vi pratat om ett begrepp som kallas för *rendez-vous*. Vad menas med detta? Förklara med hjälp av ett enkelt exempel. *(2p)*

4. Ett objekt av typen `StringHolder` (se nedan) håller reda på en referens till en sträng. Metoden `len()` returnerar längden på den aktuella strängen eller, om referensen är `null`, värdet -1. Med andra ord är tanken att det ska vara säkert att anropa metoden `len()` oavsett om referensen är `null` eller inte.

Metoderna `set()` och `len()` kan anropas från olika trådar. Referensen `s` är deklarerad `volatile` vilket innebär att trådarna inte cachar värdet utan en tilldelning av `s` blir omedelbart synligt för andra trådar.

```
1  public class StringHolder {
2
3      private volatile String s;    // can be null
4
5      /**
6       * Set the current string to str.
7       * The parameter str may be null.
8       */
9      public void set(String str) {
10         s = str;
11     }
12
13     /**
14      * Returns the length of the current string,
15      * or -1 if the current string reference is null.
16      */
17     public int len() {
18         return (s != null) ? s.length() : -1;
19     }
20 }
```

Klassen `StringHolder` antogs vara felfri och användes i ett system som körde utan problem under en lång tid. Men en dag, efter flera års användning, kraschade programmet plötsligt med följande felmeddelande:

```
Exception in thread "main" java.lang.NullPointerException:
  Cannot invoke "String.length()" because "this.s" is null
      at StringHolder.len(StringHolder.java:18)
      ...
```

Radnumret (18) refererar till `return`-satsen i operationen `len()`.

a) Hur förklarar du att `s` kunde ha värdet `null` när `s.length()` anropades trots att det finns en föregående test som ska garantera att detta inte är fallet tidigare på samma rad? *(1p)*

b) Skriv om operationen `len()` så att felet inte längre kan uppstå! Du får bara ändra på operationen `len()`. Både operationen `set()` och variabeln `s` ska lämnas helt oförändrade. *(2p)*

5. CountDownLatch

En `CountDownLatch` kan vara användbar för signalering mellan trådar – särskilt när en eller flera trådar väntar på att något ska ha hänt ett visst antal gånger. En `CountDownLatch` är inte lika generell som till exempel en monitor eller en semafor, men i vissa applikationer kan den med fördel användas i stället för dessa.

Standardpaketet `java.util.concurrent` i Java innehåller en klass `CountDownLatch`. Dokumentationen för den klassen säger att en `CountDownLatch` är:

```
"A synchronization aid that allows one or more threads to wait until a set of
operations being performed in other threads completes. A CountDownLatch
is initialized with a given count. The await methods block until the current
count reaches zero due to invocations of the countDown() method, after
which all waiting threads are released and any subsequent invocations of
await return immediately. This is a one-shot phenomenon - the count cannot
be reset.

A CountDownLatch is a versatile synchronization tool and can be used for a
number of purposes. A CountDownLatch initialized with a count of one serves
as a simple on/off latch, or gate: all threads invoking await wait at the gate
until it is opened by a thread invoking countDown(). A CountDownLatch initialized
to N can be used to make one thread wait until N threads have completed some
action, or some action has been completed N times.

A useful property of a CountDownLatch is that it doesn't require that threads
calling countDown() wait for the count to reach zero before proceeding, it
simply prevents any thread from proceeding past an await until all threads
could pass."
```

Uppgift

Implementera din egen `CountDownLatch` enligt specifikationen nedan. Du får naturligtvis inte använda Javas standardklass `java.util.concurrent.CountDownLatch` i din lösning. Använd Javas inbyggda monitorbegrepp för dina synkroniseringsbehov.

Specifikationen nedan avviker något från Javas standardklass `java.util.concurrent.CountDownLatch`, men skillnaderna är oväsentliga i vårt fall.

```
1  public class CountDownLatch {
2
3      /**
4       * Constructs a CountDownLatch initialized with the given count.
5       *
6       * @param count  the number of times countDown() must be invoked
7       *               before threads can pass through await()
8       */
9      public CountDownLatch(int count) { ... }
10
11     /**
12      * Causes the current thread to wait until the latch has counted down to
13      * zero, unless the thread is interrupted.
14      *
15      * If the current count is zero then this method returns immediately.
16      */
17     public void await() throws InterruptedException { ... }
```

```
18      /**
19       * Causes the current thread to wait until the latch has
20       * counted down to zero, unless the thread is interrupted,
21       * or the specified waiting time elapses.
22       *
23       * If the current count is zero then this method returns immediately.
24       *
25       * @param timeout  maximal waiting time in milliseconds
26       *
27       * @return true if the count reached zero,
28       *         and false if the waiting time elapsed
29       *         before the count reached zero
30       */
31      public boolean await(long timeout) throws InterruptedException { ... }
32
33      /**
34       * Decrements the count of the latch, releasing all waiting
35       * threads if the count reaches zero.
36       */
37      public void countDown() { ... }
38  }
```

(10p)

6. **U**ppstart av bromssystem med hjälp av CountDownLatch

Styrsystemet för ABS-bromsarna för en ny bil håller på att utvecklas. Vid vart och ett av de fyra hjulen (numrerade 0 till 3) på bilen sitter en liten mikrokontroller som känner av hjulets rotation. Informationen om rotationen förs sedan över till en central dator via en fältbuss (nätverk). Baserat på informationen som skickas från de olika hjulen kan centraldatorn sedan ta beslut om optimal bromseffekt. Det mesta av mjukvaran i centraldatorn är färdigskriven, men den nyanställde utvecklaren som har fått ansvaret för att skriva koden som startar upp hela ABS-systemet har fått problem och behöver din hjälp.

För varje hjul finns det en instans av trådklassen `WheelController`. Den har ansvar för att sköta kommunikationen med mikrokontrollern som sitter vid hjulet. När tråden startar upprättar den först kontakt via fältbussen med mikrokontrollerna för det aktuella hjulet. Det görs genom att anropa en färdigskriven operation `initiateConnectionToWheel()`. Detta anrop blockerar ända tills en pålitlig förbindelse har skapats (normalt högst en (1) sekund). Om kontakt inte kan upprättas blockerar anropet för evigt. Därefter anropas en operation `wheelControlLoop()` som kontinuerligt förser centraldatorn med aktuell information om hjulet. Operationen returnerar alltså aldrig.

Vidare finns det en klass `ABSSystem` som hanterar ABS-systemet. I denna finns det, bland mycket annat, en operation `initiateABS()` som ansvarar för att starta upp de fyra kontrolltrådarna för hjulen som nämndes tidigare. Kravspecifikationen för operationen säger också att om förbindelse till alla fyra hjulen inte kunnat upprättats inom tre (3) sekunder efter anropet av operationen ska den returnera `false`. Det indikerar ett problem med ABS-systemet och då kan det övergripande systemet tända en varningslampa eller vidta annan lämplig åtgärd. Om uppstarten däremot går bra (dvs att alla fyra anropen av `initiateConnectionToWheel()` returnerat inom tre sekunder i de fyra hjulkontrolltrådarna) ska `true` returneras.

Vår unge utvecklarvän har skrivit följande kod för att lösa problemet:

```
1  class WheelController extends Thread {
2      private int wheelno;
3
4      public WheelController(int wheelno) {
5          this.wheelno = wheelno;
6      }
7      public void run() {
8          initiateConnectionToWheel(wheelno);
9          wheelControlLoop(wheelno);
10     }
11 }
12 class ABSSystem {
13     public boolean initiateABS() {
14         for (i=0;i<4;i++) {
15             new WheelController(i).start();
16         }
17         return true;
18     }
19 }
```

Som synes uppfyller inte operationen `initiateABS()` de uppställda kraven. Den returnerar alltid `true` utan att kontrollera huruvida uppstarten av de fyra hjulkontrolltrådarna gick bra eller inte.

Uppgift

Ändra i den givna koden ovan så att `initiateABS()` väntar tills *antingen* alla fyra hjulkontrolltrådarna lyckats köra sina anrop av `initiateConnectionToWheel()` varvid den returnerar `true` *eller* tre sekunder har gått efter vilket den returnerar `false`!

- Använd `CountDownLatch` från förra uppgiften för all nödvändig trådsynkronisering.
- Du får lägga till nya attribut i klasserna, nya parametrar till konstruktorerna eller nya operationer i klasserna efter behov.

(5p)